1.Suppose that we wish to implement a dynamic, open-address hash table. Why might we consider the table to be full when its load factor reaches some value ***a*** that is strictly less than 1? Describe briefly how to make insertion into a dynamic, open-address hash table run in such a way that the expected value of the amortized cost per insertion is O(1). Why is the expected value of the actual cost per insertion not necessarily O(1) for all insertions? (Exercise 17.4-1 in CLRS Textbook, page 471)

Solution:

For this question, if students can specify the main logic of these subquestions, they can get full credits. Not necessarily in such a detailed way to prove it.

A. For a static open-address hash table, note that by Theorem 11.6 or Theorem 11.7 in textbook, if the load factor α is equal to 1, the number of probes in an unsuccessful search may reach infinity. Therefore, we must assure that α is strictly less than 1.

Theorem 11.6

Given an open-address hash table with load factor $\alpha = \frac{n}{m} < 1$, the expected number of probes in an unsuccessful search is at most $\frac{1}{1-\alpha}$, assuming uniform hashing.

B. For a static open-address hash table, note that by Theorem 11.7 in textbook, as long as we can ensure that table expansion does not change the amortized worst case behavior of insertion, we are guaranteed O(1) insertion.

Theorem 11.7

Inserting an element into an open-address hash table with load factor α requires at most $\frac{1}{1-\alpha}$ probes on average, assuming uniform hashing.

To keep insertion time reasonable. Insertion into a dynamic open-address hash table can be made to run in O(1) time by expanding when $\alpha \geq 0.75$ and contracting when $\alpha \leq 0.25$.

Prove

If table is at least half full:

$$\Phi_i = \frac{8}{3} num_i - size_i$$

If table is less than half full:

$$\Phi_i = \frac{1}{2} size_i - num_i$$

The expansion factor and contraction factor of table size is 2 and $\frac{1}{2}$ respectively. When load factor is less than $1/4$, the proof of deletion operation is the same as what we talked about in class. Our goal is to prove that the amortized cost when we expand the hash table is still O(1). Here are some relationships below

$$num_i = num_{i-1} + 1$$
$$size_i = 2size_{i-1}$$
$$num_{i-1} = \frac{3}{4} size_{i-1}$$

Therefore,

$$\alpha_i = c_i + \Phi_i - \Phi_{i-1}$$
$$= (num_{i-1} + 1) + (\frac{1}{2} size_i - num_i) - (\frac{8}{3} num_{i-1} - size_{i-1})$$
$$= (num_{i-1} + 1) + 2size_{i-1} - \frac{11}{3} num_{i-1} - 1$$
$$= (num_{i-1} + 1) + \frac{8}{3} num_{i-1} - \frac{11}{3} num_{i-1} - 1$$
$$= 0$$
$$= O(1)$$

C. The expected value of the actual cost per insertion is not necessarily O(1) for all insertions because the cost of inserting the mth element into a "full" table costs O(m), since all m-1 items must be copied into the new table before the new element can be inserted.

2. Write pseudocode for RIGHT-ROTATE. (Exercise 13.2-1 in CLRS Textbook, page 313)

Solution:

```
RIGHT-ROTATE(T, y)
    x = y.left
    y.left = x.right
    if x.right != T.nil
        x.right.p = y
    x.p = y.p
    if y.p == T.nil
        T.root = x
    else if y == y.p.right
        y.p.right = x
    else y.p.left = x
    x.right = y
    y.p = x
```

(x.p means the node x's parent node.)
(y means the root of the subtree where imbalance happens. T is the tree object which is flexible.)

3. Demonstrate what happens when we insert the keys [5, 28, 19, 15, 20, 33, 12, 17, 10] into a hash table with collisions resolved by chaining. Let the table have 9 slots and let the hash function be h(k) = k mod 9.

0	
1	
2	
3	
4	
5	
6	
7	
8	

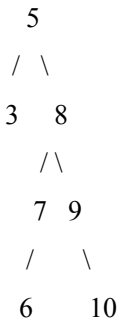
Solution:

h(k)	keys
0	
1	10 -> 19 -> 28
2	20
3	12

4	
5	5
6	33 -> 15
7	
8	17

(Reverse order is also acceptable.)

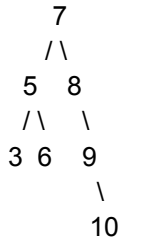
4. Building an AVL Tree out of the Binary Search Tree according to the rotation operations in the lecture. (You can simply give the final result.)



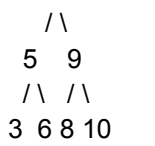
Solution:

(1) a double rotation + a single rotation

After double rotation:



Then do a single rotation:



(2) a single rotation